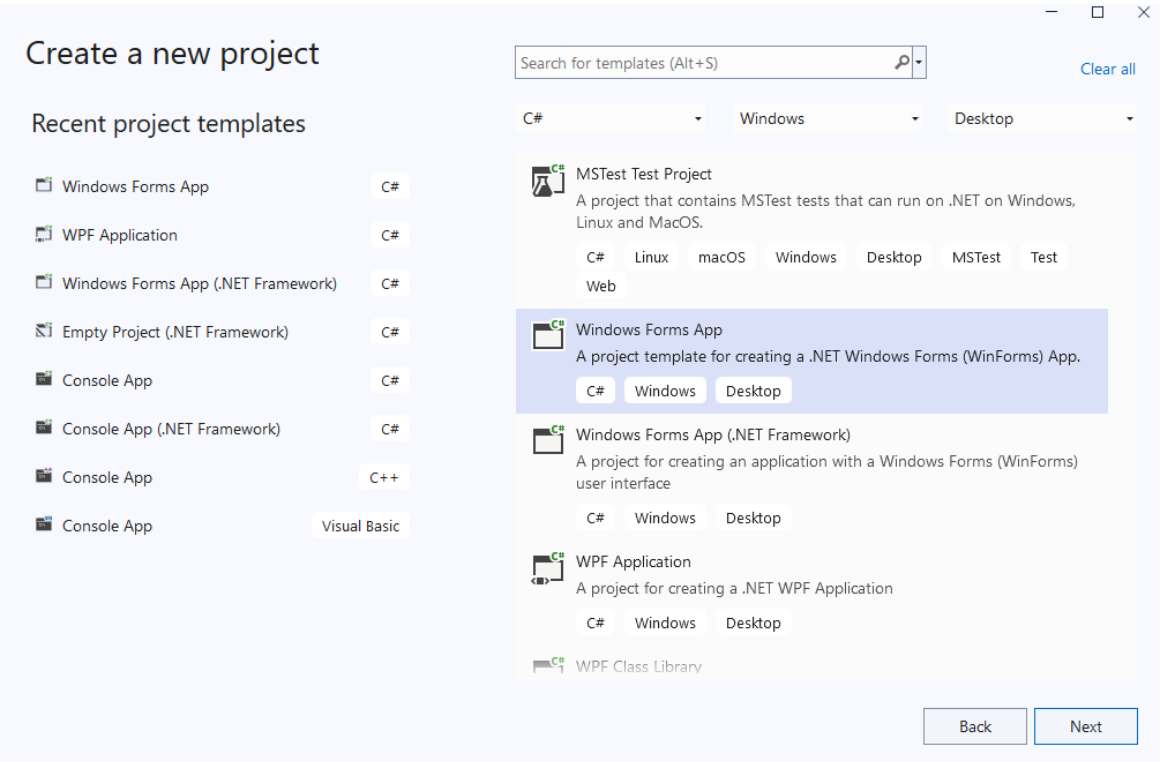
Rendezések bemutatása Form alkalmazással

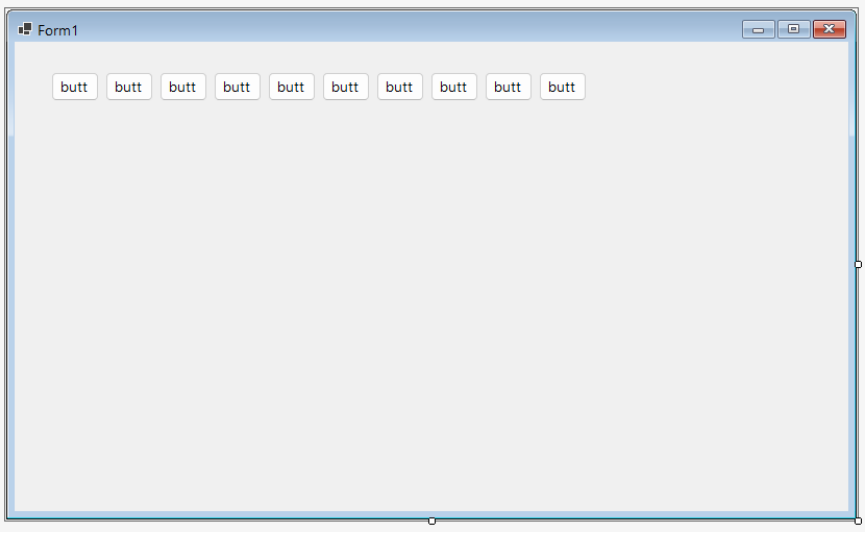
1. Az alapprogram elkészítése

Hozzunk létre a Visual Studio programban egy Windows Forms App típusú projektet:



A projekt neve legyen **RendVizu**, mappának a **D:\sajat\repo** mappát adjuk meg, a .NET 8.0 keretrendszer megfelelő. Így a projekt a reponkban lesz mentve.

A rendezés szemléltetéséhez vegyünk föl 10 Button elemet és helyezzük őket egymás mellé. A nevük alapértelmezetten **button1**, **button2**, ... **button10**. Ez legyen a gombok sorrendje balról jobbra haladva! Csökkentsük a szélességüket (egyszerre az összeset kijelölve) úgy, hogy jól elférjenek egymás mellett. Később egy-egy háromjegyű szám lesz rajtuk, tehát nem kell, hogy szélesek legyenek.



Kattintsunk duplán a Form egy üres területén, így a program indulásakor lefutó **Form1_Load**() eseménykezelőt kapjuk, ami egyelőre üres. Itt azt szeretnénk elérni, hogy a gombok a programból egy

tömb elemeiként elérhetők legyenek, mert a tömb elemeit fogjuk rendezni. Először a **Form1** { .. } osztály elejére helyezzünk el egy parancsgombokból álló 10-es tömböt:

```
public partial class Form1 : Form
{
      Button[] tomb;

      public Form1()
      {
            InitializeComponent();
      }

      private void Form1_Load(object sender, EventArgs e)
      {
            tomb = new Button[10] { button1, button2, ..., button9, button10 };
      }
}
```

A Form_Load() függvény automatikusan lefut a program indulásakor, és bejegyzi a 10 gombot a tömb elemeinek.

Térjünk vissza a vizuális felületre, tehát a Designer nézetbe! A következő lépésben helyezzünk el egy újabb parancsgombot, aminek az lesz a feladta, hogy háromjegyű véletlenszámokat tegyen az előbbi gombokra. A gomb neve (**Name**) legyen `Keveres`, a felirata (**Text**) legyen `Keverés`. Kattintsunk duplán a gombon, ezzel megkapjuk a hozzá tartozó legegyszerűbb eseményvezérlőt, tehát a gombra történő egyszeres kattintáskor lefutó programot:

```
private void Keveres_Click(object sender, EventArgs e)
{


}
```

Vegyünk föl egy véletlenszám készítésére alkalmas Random típusú elemet, és adjunk a **tomb** minden elemének egy háromjegyű számot feliratként!

```
private void Keveres_Click(object sender, EventArgs e)
{
      Random rnd = new Random();
      for (int i = 0; i < 10; i++)
      {
            tomb[i].Text = rnd.Next(100,1000).ToString();
      }
}
```

Futtassuk a programot, és a következő láthatjuk:

2. Egy cserés rendezés a gombok mozgatásával

A rendezések működését szeretnénk szemléltetni. Ha egyszerűen kicserélünk két tömbelemet (két parancsgombot), akkor a tömbben máshol helyezkednek el, de a Form-on még ugyanott maradnak. Itt a helyzetüket is meg kell változtatni, ami alapvetően a gomb **Left** és a **Top** tulajdonságán múlik. A Left az ablak bal szélétől mért távolság pixelben, míg a Top az ablak felső vonalától. Ha ezeket megváltoztatjuk, akkor a gomb helyzete is megváltozik. Például 10 pixellel jobbra toljuk a 3-as indexű gombot:

```
tomb[3].Left += 10;
```

Ha azt szeretnénk, hogy a gombok mozgása szemmel követhető legyen, akkor a mozgást több lépésben kell végeznünk, tehát pl. vízszintes mozgatáskor a Left értékét több lépésben kell egymás után módosítani, és közben várni. Várakozni a program(szál) altatásával lehet:

```
Thread.Sleep(100);
```

Ez a parancs 100/1000 = 0,1 másodpercig altatja a programot, és utána hajtja végre a következő utasítást.

A rendezések jó részében az elemek cseréje történik. Ehhez készítettünk egy **Csere(int i, int j)** függvényt a konzolos alkalmazásban, ami megcseréli az i-edik és j-edik tömbelemet. Készítsük el ezt a programrészt a Form alkalmazásban is! Vegyük föl a Form1 osztályba:

```
private void Csere(int i, int j)
{
      Button ment = tomb[i];
      tomb[i] = tomb[j];
      tomb[j] = ment;
}
```

Ez rendben, de a mozgatást is meg kell oldanunk. Bővítsük ezt a függvényt úgy, hogy a két mozgatandó gombot a többi gomb elé helyezi, hogy mozgás során előttük haladjanak, ne mögöttük.

```
      tomb[i].BringToFront();
      tomb[j].BringToFront();
```

A mozgatás úgy fog történni, hogy megjegyezzük két változóban, hogy a két gomb most hol van, mert ide kell a másik gombot eljuttatni.

```
int iLeft = tomb[i].Left;
int jLeft = tomb[j].Left;
```

Ezután felveszünk egy while() ciklust, amig addig megy, amíg az egyik gomb elég közel nem lesz a megfelelő helyéhez, és közben folyamatosan elmozdítja a két gombot:

```
while(Math.Abs(tomb[i].Left-jLeft)>10)
{
    if( iLeft<jLeft ) {
        tomb[i].Left += 10;
        tomb[j].Left -= 10;
    }
    else
    {
        tomb[i].Left -= 10;
        tomb[j].Left += 10;
    }
    Refresh();
    Thread.Sleep(100);
}
```

Tehát amíg 10 pixelnél távolabb van az i-edik gomb a j-edik gomb eredeti helyétől, addig megnézzük, hogy melyik irányba kell mozgatni őket, és elmozdítjuk. A `Refresh()` parancsra azért van szükség, hogy a Windows rendszer frissítse a Form kinézetét, tehát lehessen látni a mozgást. A ciklus utolsó lépése a várakozás 0,1 másodpercig.

De az elemek így nem feltétlenül kerülnek jó helyre, csak 10-nél kisebb távolságra vannak a végső helyüktől. Egyszerűen tegyük a helyükre őket:

```
tomb[j].Left = iLeft;
tomb[i].Left = jLeft;
```

Végül vegyünk fel egy parancsgombot a Buborék rendezés elvégzéséhez (ez volt a feladatsorban az U rendezés). Nevezzük el a gombot Buborek néven, a feliratának adjuk meg a Buborék szöveget! Rendeljük a kattintás eseményeként a korábban elkészített buborék rendezést!

```
private void Buborek_Click(object sender, EventArgs e)
{
    for (int meddig = 10-1; meddig >= 0; meddig--)
    {
        for (int mutato = 0; mutato < meddig; mutato++)
        {
            if (tomb[mutato].Text.CompareTo(tomb[mutato + 1].Text) > 0)
            {
                Csere(mutato, mutato + 1);
            }
        }
    }
}
```

Teszteljük a kész programot!

A következő oldalon jönnek az önállóan megoldandó feladatok:

3. A közvetlen cserés rendezés (K) szemléltetése

Valósítsuk meg a K rendezést, amely a Buborék-rendezéshez hasonlóan az elemek összehasonlítása után közvetlenül kicseréli az elemeket! Tegyünk föl egy parancsgombot, adjuk meg a KozvCsere nevet és a Közvetlen Cserés feliratot, és a megnyomására hajtsuk végre a választott rendezést!

4. A minimum-kiválasztásos rendezés (M) szemléltetése

Ebben a rendezésben úgy járjunk el, a gombok háttérszínének megváltoztatásával jelezzük, hogy melyik elemeket hasonlítjuk össze, és melyik a kiválasztott minimum. Ezt követően itt is elemek cseréje történik, tehát az animációval működő Csere() függvényt tudjuk használni. Természetesen itt is vegyünk föl parancsgombot, nevezzük el MinKiv néven, adjuk meg feliratként a Minimum-kiválasztás szót, és a kattintás eseményére történjen a rendezés!

5. A beszúrásos rendezés (S)

Az előző rendezésekhez hasonlóan járjunk el itt is. Annyival összetettebb a feladat, hogy nem cserék történnek, tehát a Csere() függvény helyett igazából egy másik, Eltol(int i) függvény kellene, ami az i-edik gombot az i+1-edik gomb helyére tolja. Ezenkívül a kiemelt elemet a Formon látható módon kiemeli a sorozatból, például lejjebb vagy feljebb tolja, és a végén a megfelelő helyre mozgatja.